The changing relevance of the TLB

Jessica R. Jones, James H. Davenport and Russell Bradford

University of Bath
Bath, BA2 7AY, U.K.
Email: {J.R.Jones,J.H.Davenport,R.J.Bradford}@bath.ac.uk

***Abstract*—A little over a decade ago, Goto and van de Geijn wrote about the importance of the treatment of the translation lookaside buffer (TLB) on the performance of matrix multiplication [1]. Crucially, they did not say how important, nor did they provide results that would allow the reader to make his own judgement. In this paper, we revisit their work and look at the effect on the performance of their algorithm when built with different assumed data TLB sizes. Results on three different processors, one relatively modern, two contemporary with Goto and van de Geijn's writings ([1] and [2]), are examined and compared within a real-world context. Our findings show that, although important when aiming for a place in the TOP500 [3] list, these features have little practical effect on the architectures we have chosen. We conclude, then, that the importance of the various factors, which must be taken into account when tuning matrix multiplication (GEMM, the heart of the High Performance LINPACK benchmark, and hence of the TOP500 table), differ dramatically relative to one another on different processors.**

I. Introduction

While memory hierarchies in modern processors are often discussed, and every "fact sheet" will specify the sizes (even if not other key factors) of the various levels of caches, much less attention is paid to the translation lookaside buffer (TLB). We refer the reader to [1, Figure 1 and discussion] for a good description of the TLB, though since this was published two-level TLBs and support for large pages have further complicated the situation beyond their "New Model".

In 2002, Goto and van de Geijn wrote a report [1] stating that the superior performance of their Basic Linear Algebra Subprograms (BLAS) library over competing libraries was due in part to the way that their algorithm treated the TLB: More specifically

> by casting the matrix multiplication in terms of an inner kernel that performs the operation $C = \hat{A}^T B + C$, where $\hat{A}$ fills most of memory addressable by the TLB table and C and B are computed a few columns at a time [TLB miss effect is reduced].

This was later repeated in another publication by the same authors in 2008 [2]. In neither paper was a context provided, or the importance of the TLB quantified. Perhaps as a consequence of this, some in the community were skeptical, while many simply accepted the statement. After all, the GotoBLAS library delivered very good High Performance LINPACK (HPL) benchmark results on most systems at the time, including the one at the University of Bath, and continued to do so until work on the library stopped.

To investigate the continued relevance of this claim, and the specific claim that having $\hat{A}$ filling most of the memory addressable by the TLB is important, we took a version of GotoBLAS and a system that were contemporary with the later paper's publication[2] (though post-dating [1]). On the later system the last published GotoBLAS2 was used to ensure that a BLAS kernel had been written for it. This is the last version of the library to be written and maintained by Kazushige Goto.

We altered this software to assume a different data TLB size to that on the target machine, overriding its deduction by the build system and thus the tile sizes used in the GEMV BLAS kernels selected when the library was built. It should be noted that the library builds on GEMV to produce GEMM, which is not an uncommon practice. The results of successive HPL benchmark runs were plotted and compared, and these are presented and discussed in this paper.

HPL was chosen due to its dependence on the GEMM algorithm. It was considered that if the changes to the treatment of the TLB have as great an impact as is suggested by Goto and van de Geijn, it should be visible in the results of the benchmark.

Studies such as [4] use microbenchmarks that are very similar to the GEMV loop in GotoBLAS, and these are modified in a similar fashion in order to measure the performance impact of the data (or combined) TLB on a particular machine. One downside of these microbenchmarks is that they do not present a real world context to the reader and so make judging the effect on real programs difficult. In contrast, the HPL benchmark demonstrates the value of Goto and van de Geijn's approach in a way that is familiar and easy to understand.

The production systems used were single, homogeneous nodes in clusters hosted at the Universities of Bath and Southampton. The same tests were run on Intel®Westmere processors, and also on Intel®Harpertown (Penryn), a processor that was current when [2] was published. This allowed us to run a large number of similar tests, but still with problem sizes large enough to use the GEMM algorithm designed to operate on matrix tiles within the processor cache.

We also would have liked to have compared the results from their AMD equivalents, but we do not have access to an AMD Barcelona system, and the more recent AMD Interlagos family of processors are not supported by GotoBLAS (or at the time of the writing of this paper, by its successor, OpenBLAS[5]). It should be noted that Goto and van de Geijn used a single-node Intel®Northwood system for their experiments in [1]. This

chip is now a museum piece, but a family museum has yielded one, and we have some results for it. While not being the exact same chip that was used, it is of the same microarchitecture and vintage, and should therefore show similar behaviour. Sadly, in [2] an Intel®Prescott chip was used, which, no longer being available, can only be speculated about here.

II. Background

In 2007, the University of Bath purchased a modest (9 TFLOPS) supercomputer. This machine is still in use today, and, as with most supercomputers of its size, this machine is a x86-based cluster running a variant of the Linux operating system. During the acceptance testing, one of the benchmarks on which acceptance depended was the High Performance LINPACK (HPL) benchmark.

The HPL benchmark results are highly dependent on a library known as the Basic Linear Algebra Subprograms (BLAS). In particular, on one particular subroutine: DGEMM, responsible for double precision matrix multiplication. As HPL is so important in determining the results for the TOP500 list [3] and also very commonly employed during acceptance testing, a lot of effort has been put into the BLAS and particularly into the DGEMM implementation.

For Bath's acceptance testing, the engineer tasked with running the HPL benchmark chose to use GotoBLAS. It had been stated that the cluster must achieve 80% of R_{peak} when HPL was run on it in order to pass that part of the acceptance testing.

Unfortunately for the engineer, the version of GotoBLAS installed on the Bath machine misidentified the processor as being from a different, older generation. The compute nodes on this machine all contain Intel®Harpertown (Penryn) chips, but the library's build system identified the processors as being Intel®Prescott chips. The result was dramatic. Rather than the expected 80% of theoretical peak performance being achieved, HPL only managed 50%, and no tweaking of the input parameters would allow the engineer to achieve more than a few percent improvement until the library was rebuilt for the Penryn microarchitecture.

This massive change in performance piqued our interest, and in looking at this issue several of Goto and van de Geijn's papers were examined for clues, in addition to the source code.

The two leading non-commercial BLAS libraries were, and still are, GotoBLAS[2] (forked since into several versions, the most popular arguably being OpenBLAS[5]), written by Kazushige Goto and Robert van de Geijn, and ATLAS [6], a largely auto-tuning BLAS library authored mainly by R. Clint Whaley, who notably did not give the TLB more than a passing mention in his papers. Whaley instead states that TLB problems are eliminated by careful structuring of the data, which is done to ensure contiguous access and thus promote good cache usage. This is a point mentioned also by Goto and van de Geijn, but which is accompanied by frequent comments about the importance of and thought that must be given to the treatment of the TLB. Unfortunately this importance is neither quantified nor demonstrated in any of their publications, leaving the reader to decide for himself.

One of these comments stresses the importance of avoiding a TLB miss over a cache miss, since a TLB miss will cause the processor to stall while the required data is discovered and the appropriate entry added, whereas a cache miss can sometimes be hidden by careful prefetching. Note also that, for every cache access (be it instruction or data), the TLB must be accessed first, putting it in the critical path.

With no further information to go on than the publications of rival authors, it was necessary to do some testing of our own to determine just how important the TLB is. As Goto and van de Geijn claimed that it is so essential, we decided to use the GotoBLAS library for these tests, since it seemed likely that this library would be affected, its authors having thought it necessary to stress the point in their writing. Indeed, within the build system of GotoBLAS there are two variables that are set for each supported processor microarchitecture that refer to the data TLB. These deal with the size of each entry in the data TLB and the number of entries this data TLB holds in total. Interestingly, the former variable, although set, is not used anywhere in the code.

There is a curious interaction between the data TLB and the HPL benchmark that does not seem to have been observed before. If we have differently-declared matrices, say

```
double a[1000][1000],*b,*c;
b=calloc(1000*1000,sizeof(double));
c=calloc(1000*1000,sizeof(double));
```

then accessing `b` after accessing `a`, or `c` after `b`, will need to access different *addresses*, and hence need new entries in the TLB. However, if we `free b` before allocating `c`, it is conceivable that `b` and `c` will occupy the same addresses, and hence the same TLB entries. Experimental observation of the HPL benchmark shows that, although various different matrices are solved, they are in fact all at the *same addresses*. Hence the HPL benchmark is perhaps not as much an exercise of TLBs as it might appear.

III. The Experiments

Beginning with essentially the same toolkit as was used for the Bath benchmarking exercise (that is, HPL, GotoBLAS, OpenMPI[7] and GNU GCC[8]), the getarch.c file within the GotoBLAS source code was altered to provide several new processor microarchitecture definitions based on those that would usually be used on the target machines. Each of these definitions was essentially identical to the original, except for the value of of the variable DTB_ENTRIES, which describes the size of the data TLB (DTLB) on our target processor in number of entries. (Another variable, DTB_SIZE, describes the size of each of those entries, but is not used anywhere in the code.)

The DTB_ENTRIES variable is used by the build system to define macros within the source, and affects the sizes of various things, in particular the amount by which a loop iterator within GEMV is incremented by.

This variable was altered by 25%, supplying values at 25% of the original value, then 50%, 75% and so on up to 200%. After that the step change was increased to 50% up to 800% of the original size to determine whether or not excessively larger values had any effect. The effect of this change on the compiled library was verified both by eye and using the UNIX `diff` program.

The library was rebuilt for each of the new definitions, producing a version for each modification. For speed, HPL was built to link dynamically against the BLAS library, and the LD_LIBRARY_PATH variable changed for each run to reference the relevant library. Two sets of experiments were run 30 times on each machine to gauge the effect of altering the variable on the performance of HPL on that system.

As only a single node was being used each time, and so that the same problem size could be used on both the Bath (Harpertown/Penryn) and Southampton (Westmere) machines, the N, NB, P and Q variables were chosen to be large enough to invoke the main, in-cache GEMM algorithm, but small enough for runs to complete within a relatively short time frame. In addition, HPL was configured to try combinations of all three panel factorisation algorithm variants.

On our family museum piece (Northwood), due to the (to modern eyes) rather small amount of memory available (a mere 756MB), the problem size, N, had to be reduced to 5000, half that used on the other machines. In addition, having gained access to this machine very late in our investigations, runs were made only for assumed DTLB sizes between 24% and 250% of its actual size, since previous experiments on the other two machines had shown this interval to be of the most interest.

Before we began, we verified that we would not be transparently using large TLB page sizes on any of our machines. Being production systems, kernel updates are not frequently applied, and fortunately both Aquila and Iridis are still running Linux kernel versions which pre-date this feature. On our Northwood system, we were careful to choose a similarly old kernel, although it should not be necessary.

We also checked, by eye, that the original numbers in the GotoBLAS build system agreed with the output of `cpuid` in each case.

A. Results

Figure 1 shows the results from all three machines on the same graphs, so that they may be compared at the same scale relative to one another. We include this for interest only, as we are really interested in how the performance on a single machine is affected, not how one performs in comparison to another.

At this scale, it is clear that the changes to assumed DTLB size make very little difference, the lines, practically speaking, showing little variation and being essentially flat. (It should be noted that the x axis refers to the DTLB size of that particular processor, so 100% is the point at which the GotoBLAS built system is told to assume the correct DTLB size for that specific processor.)

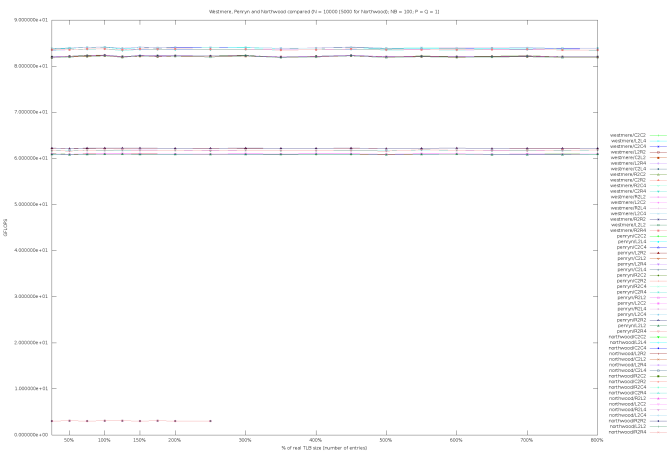

Fig. 1: The effect of changing DTB_ENTRIES on HPL on all three processors, for comparison

Considerably more marked change occurs when the HPL variables are changed in the input file, suggesting that the choice of panel factorisation algorithms, matrix size and block size are vastly more important in obtaining the best possible performance on a particular machine. Even at this scale, it is clear that the panel factorisation algorithm choice shows a clear divide on the later two processors, with results on Westmere and Harpertown dividing in each case into two distinct groups.

Although we are not so interested in inter-machine comparisons here, the results show just how significant the change in performance is when moving to a more modern machine. The Intel®Harpertown (Penryn) processors, with 4 cores, have the highest clock speed of all our systems, yet are still outperformed by the 6 core Intel®Westmere chips. The higher core count, higher memory bandwidth, faster memory and improved SIMD instructions, along with all the other improvements we have come to take for granted in modern computer systems, all contribute to the increased performance of the Intel®Westmere chips.

The single-core, single-socket Intel®Pentium 4 (Northwood) system, which predates SSE3, is comprehensively outperformed, even accounting for the loss of performance due to the smaller (halved) problem size. It has a much smaller cache, and fewer cache levels, than the other two processors, which, in addition to the myriad other improvements across the whole machine, is already known to make a noticeable difference to performance.

1) Intel®Westmere: The Intel®Westmere chip was the newest that we were able to examine in this study. It sports several improvements over the older two chips, including a 12MB Intel® Smart Cache, a dual QPI BUS and SSE4.2. This gives it a noticeable, and not unexpected, improvement in performance over even the higher clocked Intel®Harpertown processor. The 4-way DTLB on this processor supports 64 entries for 4K pages, or 32 entries for 2M/4M pages.

The results show several interesting features. See figure

2. The first observation is the similar shape to those for Harpertown (Penryn). Indeed, given the shape of the mean even between the different HPL variations (C2C2, C2L4, etc), it seems unreasonable to presume that they might be caused by noise, and MATLAB's ttest2 confirms that chance would be an unlikely cause for the fluctuations in both these and the Harpertown (Penryn) results ($P = 0.0002$).

Of most interest is perhaps the section between 50% and 150%, where there is a noticeable increase in performance up to 100%, followed by a sudden drop in performance. The change in performance is relatively small, being in the order of 0.3% of overall performance for this problem size, and is still vastly dwarfed by the effect of panel factorisation algorithm choice, not to mention other variables (particularly N and NB) that might have been chosen differently had our intention been to approach R_{peak}.

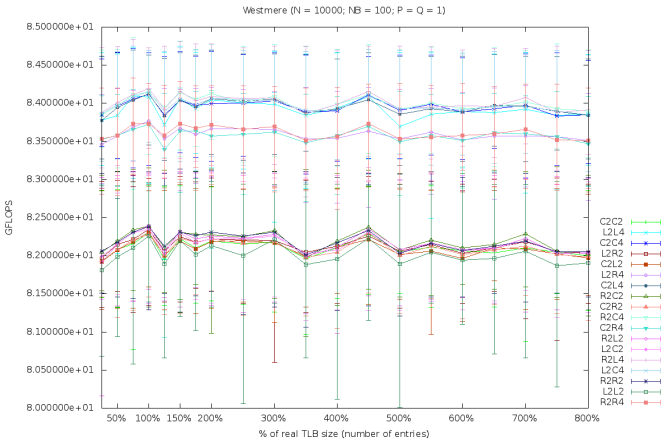

Fig. 2: The effect of changing DTB_ENTRIES on HPL on Intel®Westmere

Unfortunately a small number of our runs on the University of Southampton machine, Iridis 3+, showed up nodes performing at only 2/3rds the speed of their identically equipped peers. Their results have been discarded on the grounds of being affected by a probable hardware fault. Thus there are slightly fewer than the intended 30 runs to be considered for this chip.

2) Intel®Harpertown (Penryn): The results on this chip (see figure 3) show a strange dip when DTB_ENTRIES was altered to be 50% of the actual DTLB size of 256 entries for 4K pages. Performance overall is relatively flat. The same repetition in fluctuations can be seen across the various HPL tests, as we saw with Intel®Westmere, and ttest2 again confirmed our perception that these could not be attributed to simple noise.

A further observation is that the choice of the two panel factorisation algorithms affect performance differently on Intel®Harpertown (Penryn) than on Intel®Westmere, with right-looking approaches appearing to be a better choice on this architecture.

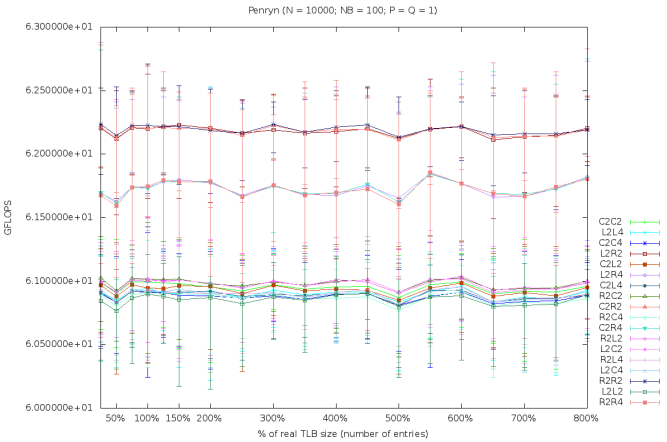

Fig. 3: The effect of changing DTB_ENTRIES on HPL on Intel®Harpertown

3) Intel®Pentium 4 (Northwood): Like Intel®Westmere, this now obsolete processor sports 64 DTLB entries for 4K pages. It supports SSE2, but none of the later improvements, and predates SMT. It is also the only 32-bit processor in this paper.

Figure 4 is perhaps the most interesting, especially when compared with figures 3 and 2. Unlike the results for the later processors, there is very little difference between the different choices in panel factorisation algorithm.

We speculate that this is due to the slightly different treatment of this processor. Since the version of the library that we are using post-dates [1], it seems highly likely that the authors will have included any changes recommended by their findings. In fact, on examining the source for the GotoBLAS library, it is clear that there are several places where the kernels differ if built for the Intel®Pentium 4 family.

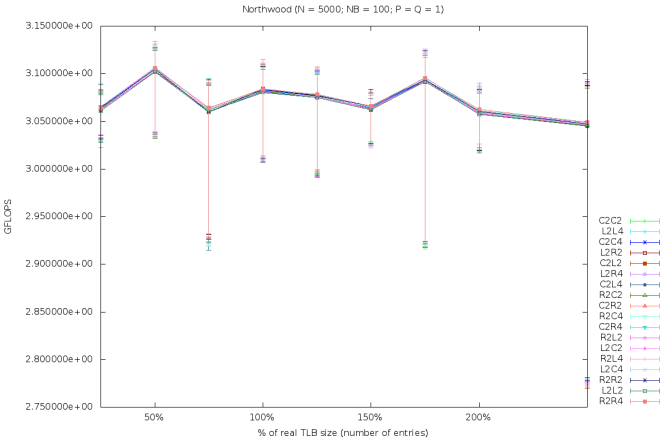

Fig. 4: The effect of changing DTB_ENTRIES on HPL on Intel®Pentium 4

IV. Conclusion

The results show that any changes to the DTLB size assumed by Goto and van de Geijn's build system result in

performance changes so small as to be, while statistically significant, practically insignificant on the processors to which we had access, being almost indistinguishable from noise to the unaided eye. It is clear that changes to the HPL input values and thus the problem shape and size affect performance considerably more noticeably, as does the choice of panel factorisation algorithms; an area we should like to investigate further.

However, in a situation where every advantage, however small, is being exploited by national facilities and companies competing for a place in the TOP500 [3] list, the performance effected by their approach to the DTLB starts to become more interesting. We suspect that the importance of a specific context may help to explain why Goto and van de Geijn appear to disagree with others in the community.

Given the nature of the algorithm employed by Goto and van de Geijn, we suspect that the main reason for their improved performance over many of their competitors is due to the cache treatment, rather than their treatment of the DTLB. Their algorithms block for the largest cache level, usually the level 3 data cache on modern processors, and level 2 on older processors such as Intel®Harpertown (Penryn). In contrast, work on ATLAS has until recently focused on the level 1 cache. Other work on cache design, such as [9], has indicated that it is the outermost cache level that has the greatest effect on overall memory I/O performance.

We anticipate continuing this work work by investigating the different relative performance of the various solvers on different architectures, particularly on modern processors. This could be better measured by looking at actual TLB miss rates, via hardware counters or similar, rather than the perhaps more difficult to measure FLOP count. We did attempt to do this for the machines in this study, using both oprofile [10] and cachegrind (part of the Valgrind [11] toolset). It was not practicable to complete this work within the scope of the existing study as cachegrind was unable to process the handwritten assembler on any of the architectures we used. We anticipate that oprofile[10] will be a more suitable tool, however, the complexity of arranging root access to a production machine has delayed this work.

Another area for further work would be to revisit the effects of varying page sizes on benchmarks on modern architectures. Similar studies, such as that undertaken in [4] have been done in the past, using the SPEC benchmarks [12], as well as hand-written microbenchmarks. A similar approach could be used here, but extra care would have to be taken due to the effects of the aggressive energy saving functionality inherent in modern processor designs.

REFERENCES

[1] K. Goto and R. van de Geijn, "On reducing TLB misses in matrix multiplication," Technical Report TR02-55, Department of Computer Sciences, U. of Texas at Austin, Tech. Rep., 2002. [Online]. Available: http://www.umiacs.umd.edu/~ramani/cmsc662/Goto_vdGeijn.pdf

[2] ——, "Anatomy of high-performance matrix multiplication," *ACM Transactions on Mathematical Software (TOMS)*, vol. 34, no. 3, May 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1356052.1356053

[3] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon, "TOP500 supercomputer sites," http://www.top500.org/, 1993. [Online]. Available: http://www.top500.org/

[4] J. B. Chen, A. Borg, and N. P. Jouppi, "A simulation based study of TLB performance," in *ACM SIGARCH Computer Architecture News*, ser. ISCA '92. New York, NY, USA: ACM, 1992, pp. 114–123, ACM ID: 139708.

[5] Z. Xianyi, W. Quian, Z. Chothia, C. Shaohu, L. Wen, S. Karpinski, and M. Nolta, "OpenBLAS," http://xianyi.github.com/OpenBLAS/, 2012. [Online]. Available: http://xianyi.github.com/OpenBLAS/

[6] R. C. Whaley, J. J. Dongarra, and A. Petitet, "Automated empirical optimizations of software and the ATLAS project," *Parallel Computing*, vol. 27, no. 1-2, pp. 3–35, Jan. 2001. [Online]. Available: http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V12-42K5HNX-2&_user=126089&_coverDate=01%2F31%2F2001&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_acct=C000010279&_version=1&_urlVersion=0&_userid=126089&md5=ea5b42507705a1461309da9391db4866

[7] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004, pp. 97–104.

[8] "GNU compiler collection," http://gcc.gnu.org/, 1987. [Online]. Available: http://gcc.gnu.org

[9] S. A. Pryzbylski, *Cache and Memory Hierachy Design: A Performance Directed Approach*, 1990.

[10] J. Levon and P. Elie, "OProfile - a system profiler for linux," http://oprofile.sourceforge.net/, Aug. 2011. [Online]. Available: http://oprofile.sourceforge.net/

[11] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," *ACM Sigplan Notices*, vol. 42, no. 6, pp. 89–100, 2007.

[12] Standard Performance Evaluation Corporation, "SPEC benchmarks," http://www.spec.org/, 1988. [Online]. Available: http://www.spec.org/

APPENDIX

A. Hardware used

The experiments in this paper were produced on three different machines:

- Iridis, at the University of Southampton : Intel®E5645 (Westmere);
  dual socket, 6 cores at 2.40 GHz (CPU family 6; model 44)
- Aquila, at the University of Bath : Intel®E5462 (Harpertown/Penryn);
  dual socket, 4 cores at 2.80GHz (CPU family 6; model 23)
- Intel®Pentium(R) 4 2.0 (Northwood);
  single socket, 1 core at 2.00 GHz (CPU family 15; model 2)

B. Software used

- GNU GCC
  - 4.2.4 on Pentium 4 (Northwood)
  - 4.3.4 on Harpertown (Penryn)
  - 4.3.3 on Westmere
- GotoBLAS2 1.13 on Intel®Westmere
- GotoBLAS 1.26 on Intel®Harpertown (Penryn) and Pentium 4 (Northwood)
- HPL 2.0

C. Linux kernel versions

- Aquila (Penryn): 2.6.18-274.3.1.el5 (Scientific Linux 5.7)
- Iridis (Westmere): 2.6.18-128.7.1.el5 (RHEL 5.3)
- Northwood: 2.6.24-32-generic (Ubuntu Hardy Heron)

D. HPL.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
10000        Ns
1            # of NBs
100          NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
1            Ps
1            Qs
16.0         threshold
3            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
##### This line (no. 32) is ignored (it serves as a separator). ######
0                        Number of additional problem sizes for PTRANS
1200 10000 30000         values of N
0                        number of additional blocking sizes for PTRANS
40 9 8 13 13 20 16 32 64   values of NB
```